Real-Time Operating Systems (RTOS) 101

Real-Time System Characteristics

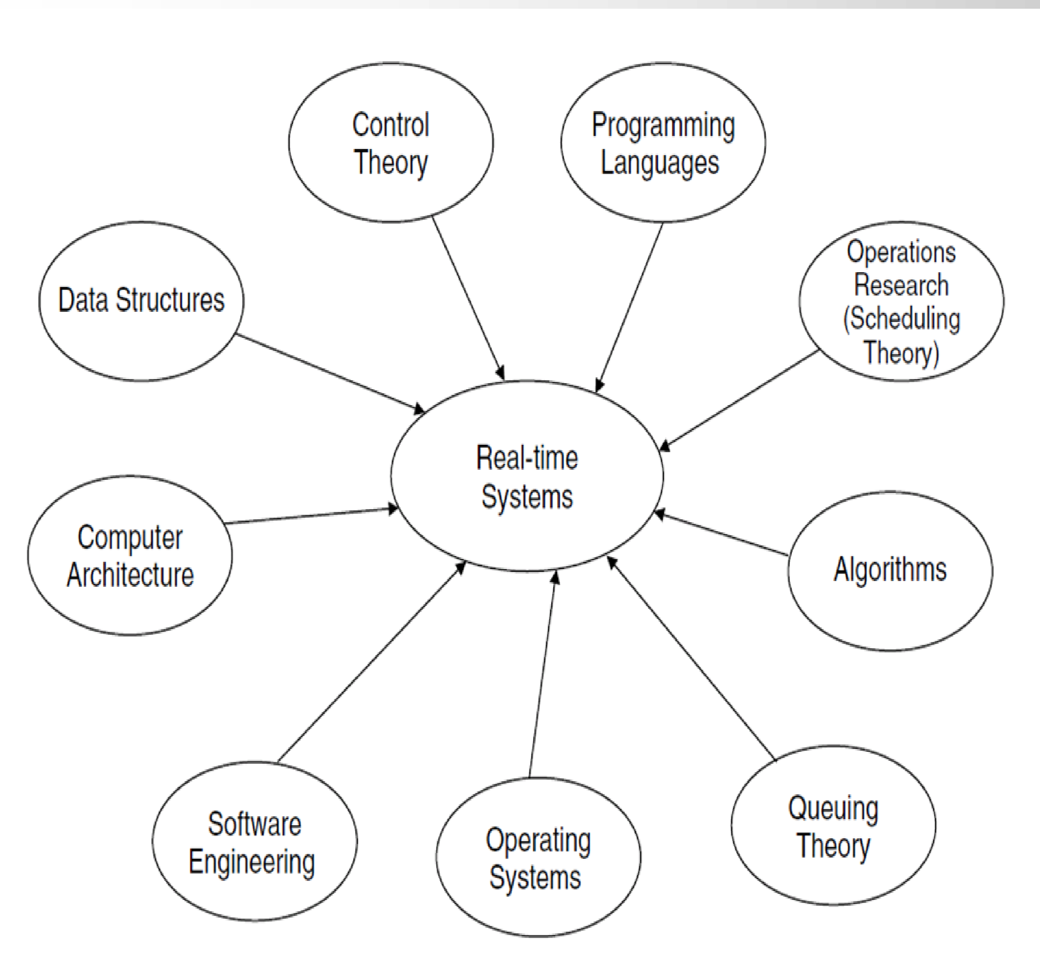
- A *real-time* system is a computer system which is required by its specification to adhere to:
  - *functional requirements* (behavior)
  - *temporal requirements* (timing constraints, deadlines)
- *Specific deterministic timing (temporal) requirements*
  - *"Deterministic"* timing means that RTOS services *consume only known and expected* amounts of *time*.
- *Small size (footprint)*

Types of Real-Time Systems

- A generic ***real-time system*** requires that results be produced within a specified *deadline period*.
- An ***embedded system*** is a computing device that is *part of* a *larger system*.
- A ***safety-critical system*** is a real-time system with *catastrophic results in case of failure*.
- A ***hard real-time system*** guarantees that real-time tasks be completed within their required deadlines. *Failure to meet a single deadline* may lead to a *critical catastrophic system failure* such as physical damage or loss of life.
- A ***firm real-time system*** *tolerates* a *low occurrence of missing a deadline*. A few missed deadlines will *not lead to total failure*, but missing more than a few may lead to complete and catastrophic system failure.
- A ***soft real-time system*** provides priority of real-time tasks over non real-time tasks. *Performance degradation* is tolerated by *failure to meet several deadline time constraints* with *decreased service quality* but *no critical consequences*.

Disciplines that Impact Real-Time Systems

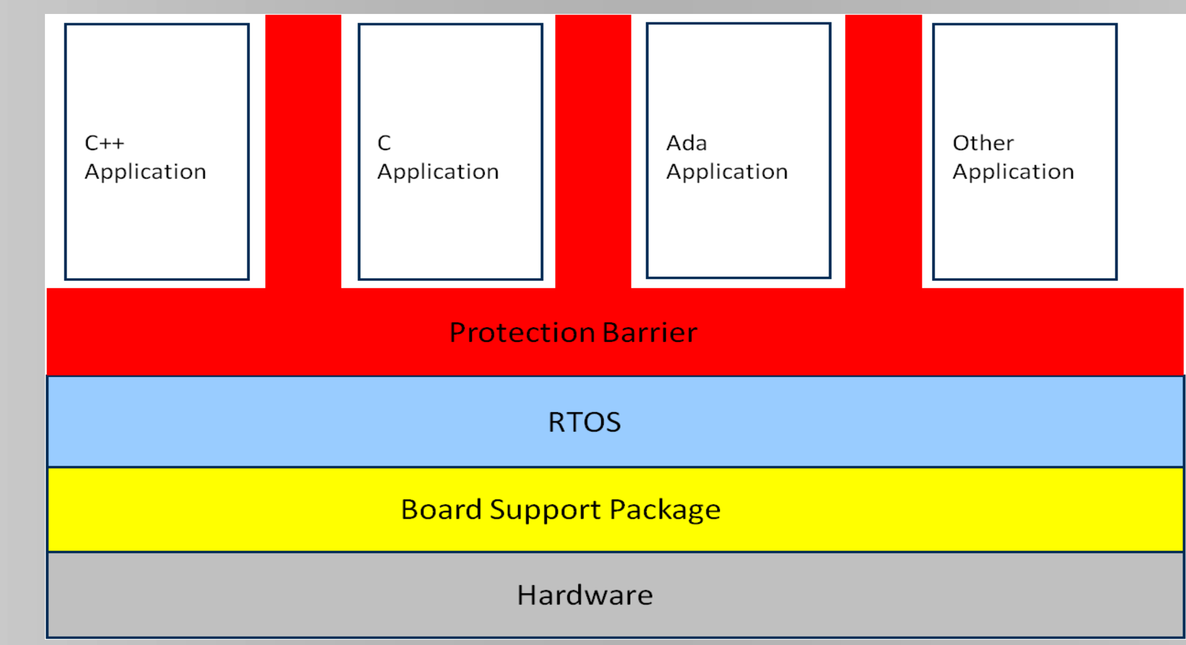
- Real-time systems engineering is so *multidisciplinary*, it stands out as a *highly specialized area*.



What is a RTOS?

- An RTOS is a *preemptive multitasking* operating system intended for real-time applications.
- It must support a *scheduling* method that *guarantees response time*
  - Especially to critical tasks
- *Tasks* must be able to be given a *priority*
  - Static or dynamic
- An RTOS has to support predictable *task synchronization* mechanisms
  - Shared memory mutexes / semaphores, etc.
- A system of *priority inheritance* has to exist
- *Manages* hardware and software *resources*.
- *Deterministic*: guarantees *task completion at a set deadline*.
  - A system is deterministic if, for each possible *state* and each *set of inputs*, a unique *set of outputs* and *next state* of the system can be determined.
- *Behavior* time constraints should be known and minimized
  - *Interrupt latency* (i.e., time from interrupt to task run)
  - Minimal task-switching time *(context switching)*

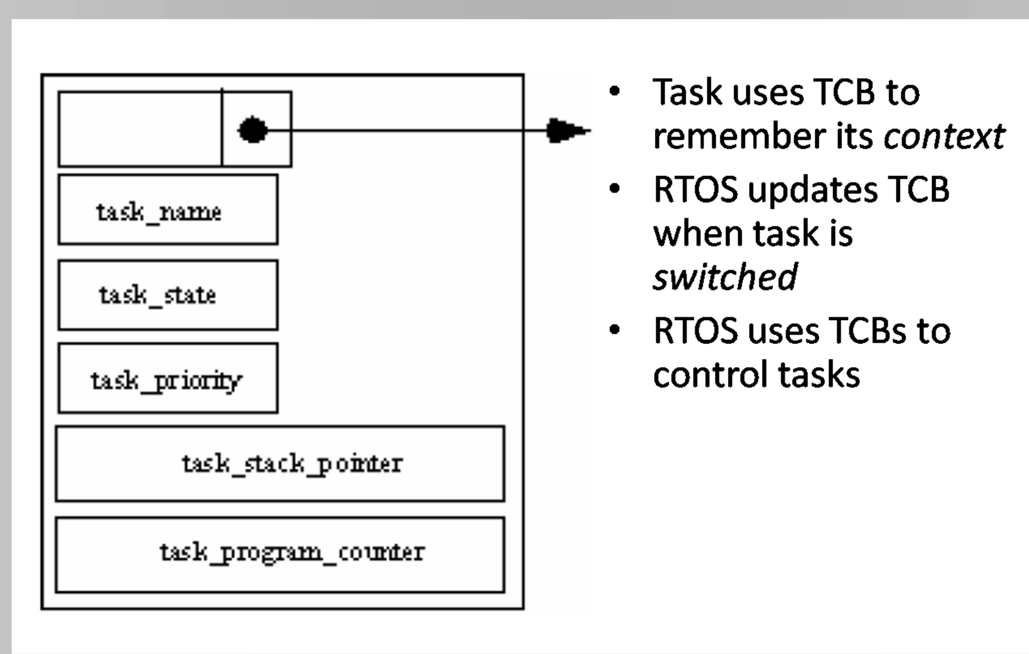
RTOS Architecture



RTOS Task Services

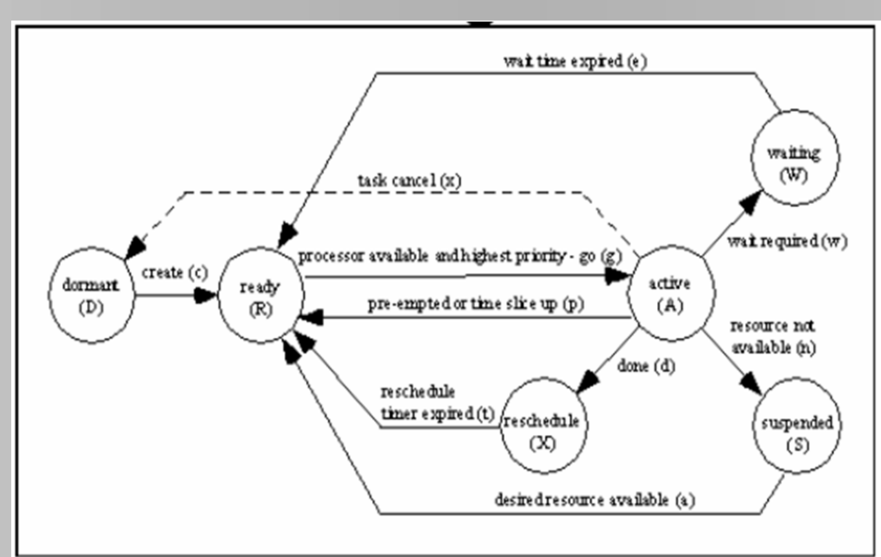
- *Scheduling and Dispatching*
- *Inter-task Communication*
- *Memory System Management*
- *Input / Output System Management*
- *Time Management & Timers*
- *Error Management*
- *Message Management*

Task Control Block (TCB)



- Task uses TCB to remember its *context*
- RTOS updates TCB when task is *switched*
- RTOS uses TCBs to control tasks

Controlling a Task



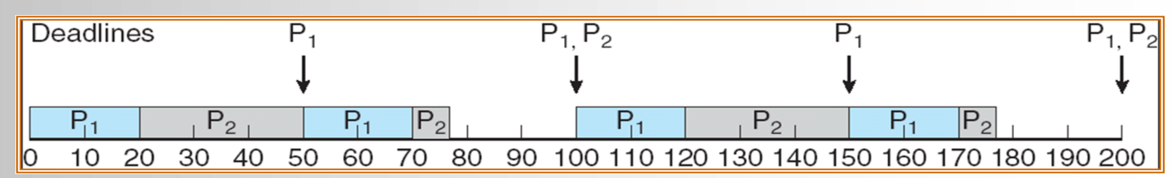
- ***dormant (idle)***: task has *no need for computer time*
- ***ready***: task is ready to go active, but *waiting* for *processor time*
- ***active (running)***: task is *executing* associated activities
- ***waiting (blocked)***: task put on *temporary hold* to allow lower priority task chance to execute
- ***suspended***: task is *waiting for resource*

Priority-Based Preemptive Scheduling

- *Problem:* Multiple tasks at the *same priority level*?
- *Solutions:*
  - Give each task a *unique priority*
  - *Time-slice* tasks at the same priority
    - Extra *context-switch overhead*
    - *No starvation* dangers at that level
  - Tasks at the same priority *never preempt* the other
    - *More efficient*
    - Still *meets deadlines* if possible

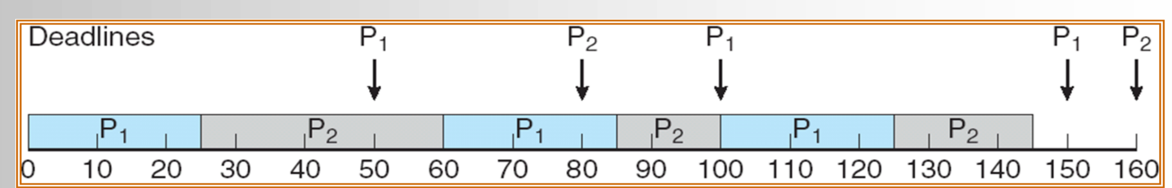
Rate Monotonic Scheduling (RMS)

- A priority is assigned based on the *inverse of its period*
  - *Shorter execution periods = higher priority*
  - *Longer execution periods = lower priority*
- Common way to *assign fixed priorities*
  - If there is a fixed-priority schedule that meets all deadlines, then RMS will produce a *feasible schedule*
- *Simple* to understand and implement
- P_1 is assigned a higher priority than P_2.



Earliest Deadline First (EDF) Scheduling

- Priorities are assigned according to *deadlines*:
  - the *earlier* the deadline, the *higher* the priority
  - the *later* the deadline, the *lower* the priority
- Priorities are *dynamically* chosen



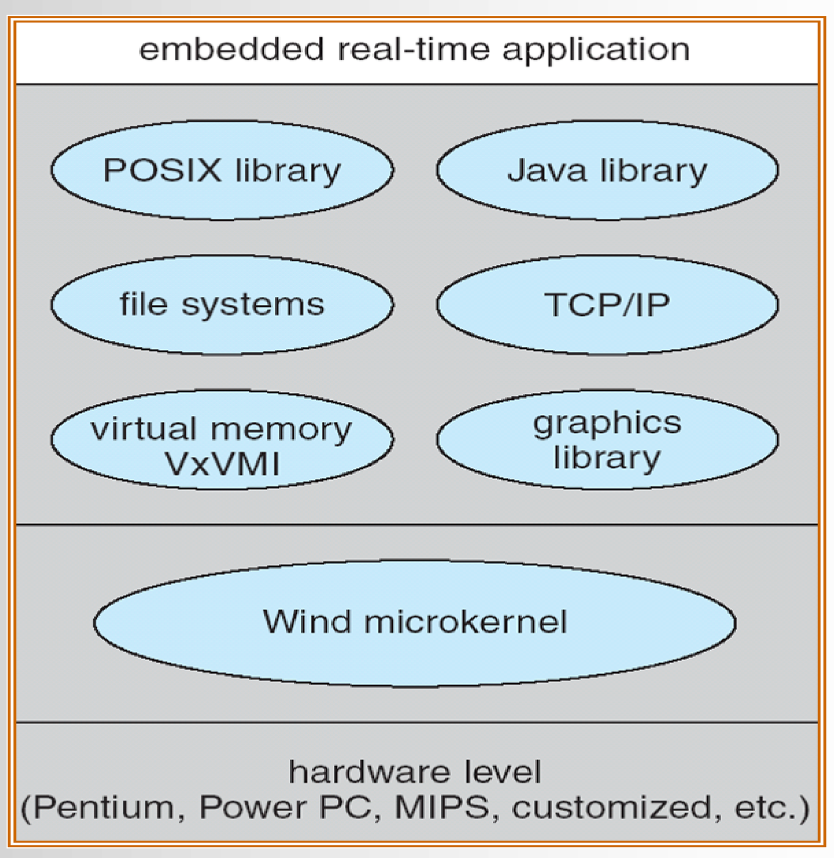
Priority Inversion

- *Lower-priority* task effectively *blocks a higher-priority* task
- Lower-priority task's *ownership of lock* prevents higher-priority task from running
- *Nasty:* makes high-priority task *runtime unpredictable!*

Priority Inheritance

- Solution to *priority inversion*
- *Temporarily increase task's priority* when it *acquires* a *lock*
- Level to increase: *highest priority* of any task that might want to acquire same lock
  - High enough to prevent it from being preempted
- *Danger:* Low-priority task *acquires lock*, gets high priority and *hogs the processor*
  - So much for RMS
- *Basic rule:* low-priority tasks should *acquire* high-priority *locks* only *briefly!*

VxWorks Architecture



Richard E. Kowalski, richard.e.kowalski@ivv.nasa.gov, TASC Inc.

NASA POC: Frank Huy, Frank.A.Huy@nasa.gov

NASA Independent Verification and Validation Facility
Fairmont, West Virginia